Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

RoboCup@Home. Commanding a service robot by natural language.

Jordi-Ysard Puigbò Llobet

Advisor/s: Dr. Cecilio Angulo Bahón

09/09/2013

Acknowledgements

I wish to thank various people for their contribution to this project; Dr. Cecilio Angulo, my advisor, for being in the right place, in the right moment, and so making possible the work on this project and providing me this opportunity to learn about robotics and about myself. To Dr. Ricardo Tellez, whose assistance and enthusiasm were clue for the development of the project and for making possible the RoboCup experience in the team Reem@IRI. I would like to express my very great appreciation to all the members in the Reem@IRI team, who participated in RoboCup and made the development of this thesis an unforgettable experience. Very special thanks are extended to Oscar for his support in moments of doubt. I would like to thank my parents, who are always in my heart, for believing in me and lending me their hand, blindly, whenever I could have needed it. I bear in mind Christian and Lydia for their support all the years of the master and all the people that has not been mentioned but have been there, listening my complaints and supporting me and my ideas and showing me the path when it wasn't clear.

Finally, my special thank to my beloved Gemma, who has spent with me the last moments of this thesis and gave me her care and attention and her support and encouragement in the lowest moments.

Contents

Chapter 1

Introduction

It was in the ancient Greece that myths were written and, among already there one could find the human desire of robotic servants. It was Hephaestus, god of technology, blacksmiths, craftsmen and artisans who is said to have built robots to help him on his workshop. This show how deep in our thoughts was this desire that one could find stories and tales of human-shaped machines that danced in china or inanimate materials like mud that gave shape to golems in Jewish tradition.

In the renaissance, a lot of automata began to arise, beginning by Leonardo Da Vinci to the artisans from China and Japan, mankind was trying to produce automatic machines, sometimes for their own benefit, some other times to their delight and fascination.

But it wasn't until the digital era that the dream began to seem feasible. After millennia of wondering of automated robots, computers showed that automatic calculus was possible and from this, ideas of an automated mind arose. Theories for cognitive architectures are born since the early stages of artificial intelligence, cognitive architectures that now are a reality.

Thanks to the technological advances and the knowledge about the mind, what once was material for fictional tales, now is feasible and only matter of time. There is a lot of research on robotics and cognition that is beginning to get coupled into what are called "service robots".

In this thesis, I present a system that participates in a competition designed for this kind of robots. A competition that have on its basis the same dream that humans have had all around the world for centuries: the cohabitation of humans and service automatons.

The next section will describe the context of this project, the RoboCup initiative, the robot Reem and the problem itself, together with the definition of the goal that is pursued within this document. Section 3 will analyse the State of the Art in humanoid robotics, service robots, semantic extraction from commands and planning and cognitive architectures for robots. In section 4, the software and strategies to solve the problem are described and argued, together with the methodologies and the implementation, in order to present the results and evaluation of the proposal in section 5, to end with a discussion on the achievement of the goals in the last section 6.

Chapter 2

Definition of the problem and goals

2.1 Robocup

2.1.1 What is Robocup?

RoboCup is an international project to promote AI, robotics and related fields. It's objective is to promote AI and intelligent robotics research by providing problems where a wide range of fields and technologies can be integrated and tested together. More information can be found at http://www.robocup.org/.

2.1.2 Origin

The year 1997 is remembered as a turning point in the history of artificial intelligence and robotics:

- IBM Deep Blue defeated that year the human Chess Master Gari Kasparov, the Chess World Champion from 1985 to 2000
- NASA's Pathfinder mission landed successfully the first autonomous robotic system on the surface of mars

Was in 1997 that the first Robot World Cup Tournament (RoboCup) was held in Nagoya. This was an international initiative born in the Workshop on Grand Challenges in Artificial Intelligence in October, 1992 in Tokyo that led in June 1993 to the Robot J-League, name that stands for the japanese soccer league. During the workshop, serious discussions on the technological and financial feasibility and the social impact of using the popular soccer game to promote science and technology. The first official RoboCup games and conference was held on 1997 with over 40 teams and over 5000 spectators.

2.1.3 Objectives

The first and main objective of the RoboCup initiative is to promote robotics and AI research by offering an appealing but not any less an [onerous] challenge. Building a robot that plays soccer is a goal that does not generate a significant social or economic impact but its accomplishment would certainly be a major achievement on the fields of AI and robotics.

For this reason, a long term goal, 50 years, was proposed for the RoboCup initiative:

> "By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."

The 50 years term is chosen because in recent history, 50 years is what took from the Wright brothers first aircraft to the Apollo mission to land a man in the moon and safely return him to the Earth. By the way, it took also near 50 years from the invention of the first digital computer to the Deep Blue defeating a Chess World Champion.

For this reason, like in the Apollo project, RoboCup initiative was posed as a Landmark Project, which beside the benefit of achieving the goal itself, which doesn't have any direct economic impact (having robots playing soccer in a professional league), the technological achievements developed to accomplish it can be significantly powerful for the development of next generation industries.

Although the accomplishment of the general goal might not be feasible in a near term with the current technologies, the subgoals that are derived from it are by themselves important achievements for the current State of the Art of both robotics and AI. The first subgoal to be accomplished in RoboCup is "to build a real and software robot soccer teams which plays reasonably well with modified rules". And even this goal will undoubtfully generate technologies that will impact a wide range of industries and research.

2.1.4 Leagues

Nowadays, RoboCup have the following leagues:

RoboCupSoccer Creating teams of fully autonomous, cooperative robots that exhibit competitive behaviors and strategies. There are different leagues, for humanoid or non-humanoid robots, to evaluate behaviours or technical skills.

RoboCupRescue Developing highly mobile, dexterous and semi-autonomous robots to perform dangerous tasks and save people during emergencies, mapping and/or negotiating in limit environments.

RoboCup@Home Creating autonomous service robots, that help people at home and in public environments by natural interaction methods.

RoboCupJunior : Motivating young people to learn skills and knowledge necessary in science, technology, engineering, and mathematics as well as to foster their soft skills through participating in the creative process of building and programming autonomous robots.

2.1.5 Robocup@Home

The objective of RoboCup@Home is to develop service robot technology with high relevance for future persona domestic applications. It is the largest international annual competition for autonomous service robots and is part of the RoboCup initiative. Every year, a set of tests are selected to evaluate the robot's abilities and performance in a realistic environment setting. This competition is focused specially in the following topics:

- Human-Robot-Interaction and Cooperation
- Navigation and Mapping in dynamic environments
- Computer Vision and Object Recognition under natural light conditions
- Object Manipulation
- Adaptive Behaviors
- Behavior Integration
- Ambient Intelligence
- Standardization and System Integration

2.1.5.1 Tests

1. *Robot Inspection*

   Each robot has to register itself and get aproval to participate in the competition.

   The test consists in:

   - Enter the arena
   - Approach the registration desk
   - Introduce themselves
   - Present the registration form
   - Exit the arena through the dessignated door
   - While leaving, the inspectors hit the emergency stop button and release it again, so that the robots leave the arena

2. *Follow me*

   This test focuses on tracking and recognizing a previously unknown person, basic interaction and signalling capabilities, and safe navigation in unknown environments and narrow spaces with other people walking around or blocking the way.

   The test consists in:

   - Memorizing the operator
   - When the robot is ready, the operator starts walking in a natural way on a designated path, while the robot must follow him dealing with diferent obstacles, within a designated time

   Obstacles :

   - Two persons block the direct passage and one of them starts walking crossing between the robot and the operator
   - The operator guides the robot into an elevator and they have to leave in reverse order
   - The operator crosses through a small crowd of people (4-5) and waits for the robot to surround the group of people and is not allowed to move back

3. *Clean up*

   The robot has to clean up a room in the apartment that is messed up with objects.

   The robot is commanded to tidy up a room The robot scores for each object found in a wrong place and is delivered in the correct location, predefined beforehand

4. *Cocktail Party*

   The robot has to learn and recognize previously unknown persons, and deliver drinks.

   The robot enters a room where 5 people are waiting. It asks who wants a drink, approaches the persons and asks them which drink do they want. The robot scores for each order detected correctly, for each object successfully grasped and for each object successfully delivered to the correct person.

5. *General Purpose Service Robot*

   This test evaluates the abilities that were required from the robot throughout the previous tests. In this test, the robot disposes of an extended period of time (30-45 minutes) to solve multiple tasks on request. This tasks are requested in natural language and the test itself has not a predefined story neither a predisposed order of tasks. The actions that are requested from the robot are generated randomly by the referees from a larger set of actions. The commands are organized in three categories that will be described with more detail in section 2.3.1.

6. *Restaurant* The robot has to access an unknown, real environment like a restaurant or a shopping mall. There will be some people at the tables. The robot must be introduced by a guide. It will be presented the tables and the places where drinks and food are placed. After this, the robot will receive three orders from the guide and it will have to bring the specific items to the specified tables.

7. *Emergency Situation*

   The robot has to properly react to an emergency situation in the house.

   The main story of this test is:

   - There is a fire in the kitchen
   - The robot has to discover the danger
   - Then search for people in the environment
   - Notify them
   - Guide them to the emergency exit

8. *Open Challenge*

   During the Open Challenge teams are encouraged to demonstrate recent research results and the best of the robots' abilities. It focuses on the demonstration of new approaches/applications, human-robot interaction and scientific value.

Table 2.1: Reem-H3 Technical Specifications [1]

REEM Technical Specifications	
Weight	90 Kg
Height	1.70 meters
Battery autonomy	8 Hours
Degrees of freedom	22
Payload	30 Kg for the mobile base
	3 Kg each arm
Speed	4 Km/h
Computer	Core 2 Duo + ATOM
Sensors	Microphone, stereo-camera, laser, ultrasounds, accelerometers and gyroscopes

2.2 The platform: Reem

Reem is a emphhumanoid service robot platform created by PAL Robotics. It was originally conceived for event assistant, bothe for guidance and entertainer, thanks to its social and service oriented capabilities. Due to its specifications it is a very suitable robot for participating in the RoboCup@Home league.

2.2.1 Technical Specifications

Table 1 shows the main characteristics of Reem-H3. Reem is a humanoid service robot with human-like size and weight. Its navigation system is based in a wheeled base instead of being biped, like the new Reem-C already is. The problems that come with biped robots are first, that the robot must preserve its balance in order to avoid falling down; and second, the robot must be able to stand or lay in a resting position when it is turned off, again, without falling down. The wheeled locomotion facilitates these navigation issues given that balance is a parameter that doesn't need to be considered. Given that the first test in the RoboCup is testing the emergency button, using a wheeled robot seems a very appropriate option. Reem has also two arms with hands provided with the capacity to grasp small objects, like cans, bottles, cereal boxes, etc. It is also supplied with some sensors like lasers, that allow the robot to map its environment and to identify obstacles; stereo cameras, for recording, 3D scenes interpretation, etc. ; sonars, specially for collision avoidance and human and robot safety issues; and accelerometers and gyroscopes, specially for pose estimations. It was also provided with a kinect sensor on a headset on her head in order to achieve a better and easier 3D representation of the graspable objects during the RoboCup days.

2.2.2 Software Specifications

Reem is compatible with OROCOS control and automation software and with the Robot Operating System ROS. The control software for the robot is mainly developed by PAL Robotics itself and it is something out of the scope of this paper.

By the way, ROS is the middle-ware platform over which most of the software is developed in the scope of this thesis. For this reason I will make a bit of introspection on what is ROS and how it works.

ROS

ROS (Robot Operating System)[2] is a software framework for robot software development. It provides standard operating system services like hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes and package management. The library is supported on Ubuntu-Linux distributions.

ROS has two essential blocks: The operating system block *ros* with the characteristics mentioned above and *ros-pkg*, which is a suite of user contributed packages that add functionalities such as planning, localization and mapping, grasping, perception, simulation, etc. ROS is dessigned to be modular, in any of the cases.

Some concepts about ROS need to be introduced for the proper understanding of the core of this thesis:

Node Nodes are processes that perform computation, or what is the same, are software modules.

Message Nodes communicate with each other using messages. A message is a a strictly typed data structure of standard primitive types (integer, floating point, boolean, etc.) or arrays or other messages.

Topic Messages are *published* by nodes into topics. Topics are only strings (like "map" or "ASR" where other nodes *subscribe* to read that topic. Nodes can publish and/or subscribe to multiple topics and they aren't aware of each other.

Service Services are special nodes defined by a string and one message for request and one for response. These services are designed for synchronous communications, avoiding the publish-subscribe model.

2.2.3 Capabilities

Reem has already integrated a whole set of abilities or functionalities that make him suitable for the participation in RoboCup@Home and the use of a cognitive architecture to perform high-level control of her actions in the role of a service robot.

Among these functionalities stand out:

Navigation and Slam Reem can go from one point to another, map his surroundings, avoiding obstacles and storing check-points.

Face Recognition Reem is capable of learning a face of a person looking directly at him and recognising him in the future.

Speech Recognition Reem has integrated speech recognition software for natural language understanding.

Object Recognition Reem has been trained with object recognition algorithms for a variety of household objects.

Grasping Reem is capable of grasping objects and bringing them to specific positions or persons.

2.3 The problem

The problem contextualized within this chapter in the previous sections is then that of allowing a robot, specifically PAL Robotics' Reem-h3 to successfully perform the tasks required for passing the General Purpose Service Robot test in the RoboCup 2013 terms and context.

Specially, the goals were provided by PAL Robotics and were:

- Considering a correct input coming from the automatic speech recognition system, develop a parser that successfully extracts the semantic knowledge from a command of the first category.
- Considering the output format of the previous parser, develop a system that dynamically chooses the best set of actions for the accomplishment of the listened command.
- Integrate these systems on the Reem-H3 platform interfaced under the middle-ware ROS.

2.3.1 General Purpose Service Robot

Within the context of the RoboCup@Home league, the General Purpose Test particularly focuses on the following aspects:

- *No predefined order of actions to carry out:* The commands are generated randomly from a set of 25 different names, 21 different objects and 18 different locations combined with 23 different action verbs, what makes the use of state machines unfeasible.
- *Increased complexity in speech recognition:* Long sentences, synonyms, missing information, co-references and erroneous information.
- *Environmental high-level reasoning:* In the first category the reasoning is merely to contextualize the whole sentence but in 2nd and 3rd categories of sentences, the commands contain either missing, erroneous or ambiguous information.
- *Robust long-term operation:* The robot will be given three commands and can't be manually operated in any moment. The robot must be able to wait for the referee, fulfil the given command and wait for another one without crashing.

2.3.2 Sentence Categories

There are three categories of commands that the team can choose for the robot, although the requirements of the thesis are just to fulfil those of *category I.*

Category I: The command is composed by three actions, which the robot has to show it has recognized. The robot my either ask for confirmation or for repeating the full command.

Category II: The robot gets a command that does not include all the information being necessary to accomplish the task. The actual commands will be underspecified by, for example: only giving the class of the object (”bring me a drink”) or location (”go to a table”), and not the actual object or location, or not providing the location (or its class). The robot can ask questions to retrieve the missing information about the task, but is not required to. Must be noticed that the robot does not know the position of the objects before-hand, although it already knows the objects and locations that belong to each category. Also the location of the objects must not be of common sense, it is completely random.

Category III: The command contains erroneous information. The robot should be able to realize such an error while trying to carry out the task, get back to the operator, and clearly state why it wasn't able to accomplish the task and, if possible, find an alternative to finish the task.

2.3.3 Semantic Challenge

The first challenge facing this problem is designing a methodology for extracting the necessary knowledge from each kind of sentence, accurately, and to retrieve it in some way that can be unambiguously interpreted by the robot.

This implies finding or defining the tools that allow the accomplishment of this purpose and integrating them with the embedded Automatic Speech Recogniser (ASR), ROS and the cognitive architecture that comes behind.

2.3.4 Adaptation Challenge

In terms of action execution, the challenge is to avoid rigid, sequential programming and develop a model capable of dynamically adapt to the current goal and available information of the world to fulfil that goal.

2.3.5 Technical Challenge

In terms of the RoboCup@Home there is also a technical challenge that is that the robot must have the ability to perform the three given commands within the required time. In this case, the technical aspects are not in the scope of this thesis and are in charge of PAL Robotics and will not be discussed any more.

Chapter 3

Related Work

3.1 Humanoid Service Robots

3.1.1 Service Robots

Between the 1960s and the '90s, most robots – and robotics in general – were related to industrial applications, mainly intending to rationalize production at manufacturing sites. Nowadays, robots are reaching exceptional capabilities and robustness and are becoming more present in our lives [3]. Service robotics is an emerging application area for human-centred technologies. The rise of household and personal assistance robots forecasts a human–robot collaborative society. This means that a lot of robots are currently being developed not only to work *for* humans but also *with* humans. A whole field known as Human-Robot Interaction (HRI) must be addressed, joint with the technical issues. For this reason, one aspect in RoboCup@Home is the safety of humans while robots are working around them and the appearance they exhibit[1].

International Federation of Robotics defines a service robot as:

> A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations. [4]

Service robots typically share the human environment and exhibit basic intelligent behaviour to accomplish assigned tasks. They are considered as a branch in the evolution of robots and widely recognized as a dominant research field in the near future [3].

[1]See section 3.1.2 for more introspection in the uncanny valley.

Recently, a rapid evolution on this so-called ”service robots” has become apparent, turning what has been for a long time a dream of humankind into reality. The research in this field is developing robotic systems that can be considered modern implementations of personal robots or robotic assistants.

But as mentioned, one important factor in the integration of service robots is the psychological aspects mentioned before, because, opposite to manufacturing robots, service robots are designed to work on uncontrolled and unprotected environments such as a house, while an accident could lead to a social non-acceptance of the service robots condition.

3.1.2 Humanoid Robots

Humanoid Robots are robots with human shape. They can be legged or wheeled, but they usually have a human-like face, in shape and characteristics and a body with two arms and a human weight and size.

Because of his anthropomorphic structure, humanoid robots are ideal service and assistant robots. They fit to the idea of a general purpose robot to perform everyday tasks in human environments. Also, lately, their capabilities have been enhanced both, thanks to the advancement in technology and the new models and theories for modeling human-like behaviours and abilities, specially in control and modeling.

ASIMO from Honda [5], WABIAN series from Waseda University [6], HRP-3P/HRP-4C from AIST/KAWADA [7, 8], Johnnie from Munich University [9] , and HUBO from KAIST [10] are representative humanoid robots, although they are not conceived for service but for research purposes.

It’s important to remark that most of the investigation in humanoid robots is being held in locomotion/mobility mediums, path planning and service oriented skills [11–16]. To say, most of the work has been on the skills needed by a service robot, but not on the reasoning and the way to have a service robot working autonomously. The aim of this thesis is to make a step further of this, but not less important, work and given a set of already implemented actions or macro-actions, hopefully robust enough, define an environment where the actions can be performed autonomously by the robot in order to fulfil some commands.

3.2 Reasoning Architectures

There is a lot of investigation in cognitive architectures. With the rapid evolution robots, the research on embodied cognition has began to grow and, for this reason, researchers on cognitive robotics have derived their efforts on semantics [17], navigation [18, 19], cooperation and multitasking [20, 21] or low-level robot control [19, 22].

Usually, the research have worked under simulations instead of directly on the robots, although some [19, 23] have implemented their architectures on robots. Specially in [23] Chen explains how they have created their own cognitive architecture instead of using one from the existing, like Soar or SS-RICS, which are the most common architectures. This specific team participated in RoboCup@Home for the last four years.

3.3 Imperative sentence parser for robots

There is some, although not much, literature referring the understanding of natural language instructions in the case of human-robot interaction (HRI) [17, 24, 25]. In all the cases, the approaches are presented to face any kind of natural interaction with humans. They all agree on that the understanding depends strongly on the environment and the abilities of the robot. This means that the representation of the knowledge strongly influences the understanding of the sentences.

Although this, they approach the problem in the context of natural HRI, in terms of making the human feel more comfortable with the robot, while in this thesis the aim is the functionality.

On the other side, usually semantic extraction has been addressed by the use of semantic dependency ontologies like WordNet[26] or verb-argument annotated corpus like PropBank[27] or VerbNet[28], like shown in [29].

Another approach, more simple is encoding layers of post-processing of relations after a dependency parsing is obtained [30]. This approach is more rigid than the previous one because the layers of post-processing are rule-based, but are faster and simpler to implement.

In any of the cases, it's important to notice that there is no best solution, but a good solution in the application context that is being faced.

Chapter 4

Methodology and strategies to solve the problem

4.1 Semantic Information Extraction: NLTK

In order to participate in the General Purpose Service Robot tests in RoboCup, the robot needed to be able to understand which command was given. In the following section a short description with examples of the kind of commands that can be found on each category is described, to continue with the decomposition of this goal into different subgoals and finish with a description of the solution proposed for this specific problem.

4.1.1 Example of Commands

As it has been explained before, there are three categories of sentences that can be given to the robot during the test. Only the first two are in the scope of this thesis.

Category I: These kind of sentences are commands composed by three different actions. These actions are connected in a single sentence with conjunctions (*'and'*), transition particles (*'then'*) or punctuation marks (*','*). Should be noticed that, given that the output comes from an Automatic Speech Recognition (*ASR*) software, all punctuation marks are omitted. Also, the pronunciation by the referees must not be forced, so no special emphasis is put on the commas to separate the commands, so the test would be more realistic.

In this category, the composing sentences correspond to specific commands. They are strongly correlated and the main concern resides in the understanding of the sentences.

Some examples would be:

> Go to the kitchen table, find a coke and grasp it.

> Follow the person in front of you, introduce yourself and then, exit the apartment.

Category II:

Sentences in Category II are more dedicated to the reasoning rather than the Natural Language Processing. Specifically, the command itself does not provide all the necessary information that would allow to execute the command directly. This means that categories of items and indefinite determinants are used while the specific location of the items involved in the test is hidden. For this reason the robot is expected to, not only understand the sentence, but realize which information is still needed in order to accomplish the proposed goal.

Also, in this case, the orders are more general. While in the first category, the example command would expressly ask the robot to go to the kitchen, then, there, to find a coke to finally command her to grasp it, in this second category the robot is only given a command like:

> Bring me a drink.

or

> Find a snack.

This means that the robot does not only have to execute a specific command, but to find a way to fulfil a concrete task that involve other subtasks (planning) and, so, to eventually find a way to obtain this missing information, either asking or looking by herself in the test arena.

Category III: Although this category is out of the scope of this thesis, it is important to consider it because it gives a specific hue to the *GPSR* test.

In this case, the sentence is of the same kind as in the second category, but some of the information will be wrong. For example, the robot will be asked to find a specific person in a room where he/she isn't or she will be told to bring a location instead of an object. This means that the robot should be robust enough to detect the information and correctly identify that the action is not possible with the given arguments.

This have been presented in this section in order to discuss this topic later in the conclusions of this thesis.

4.1.2 Objective

Given the previous description of the problem, we can identify one specific goal, in terms of the semantic extraction point, that encloses the solution to the two categories of commands described above:

> GOAL: To successfully extract and retrieve the relevant knowledge from an imperative sentence in the context of RoboCup@Home.

There are three important keywords in this goal definition:

Extract and Retrieve The software must be able to read the command, process it and return it in a format that is specific and complete for the later processing of the reasoner module.

Knowledge The information that the system needs is not specifically included in the sentence, but it is a projection to what the robot needs and understands. This means that 'leave', 'and', 'exit', 'coke' and any other word that the referee can pronounce and the robot can listen, does not necessarily mean different things, or even anything. What is needed is to filter or translate all these words into *useful information* or *knowledge.*

Imperative There is a lot of literature in parsing sentences and semantic extraction, but most of them is found to work on press articles, web pages or book text. There is not much work found on the processing of imperative sentences or commands given that this is useful only for agents that can act in our world and service robots are still in their early stages.

4.1.3 Dependency Parsing and NLP Tools

Given that the context is action-oriented and the imperative sentences are commonly expressed in the form `Verb + Complements` the use of natural language processing is mandatory, with special emphasis on dependency parsing.

NLTK

NLTK is a platform to build Python programs that work with Natural Language Processing. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning[31]. For this reason this toolkit becomes an indispensable software for this thesis, also considering that the thesis is aimed to be programmed in python.

Stanford Parser

Stanford parser is a state of the art dependency parser that uses probabilistic-Context Free Grammars (pCFGs) as the basis of the dependency parsing, combined with semantic role labeling[32]. It was trained over the well-known PennTreebank, what means that it has been trained specially for general use sentences, like in books or the press. This means that, under the probabilistic training assumption that the data must be representative of the context on which the - in this specific case - parser will have to work. But most of this corpus isn't composed by sentences in imperative mood, excluding the fact that there is no analysis of the mood, what seems an interesting option in cases like this one. Also, imperative sentences are usually short and doesn't have many subordinated propositions. From this, one could imagine that for large complex sentences like those described in RoboCup@Home, there is a high probability that the parser tries to assign them non-imperative roles leading to parsing errors.

Stanford Parser is constructed as an unlexicalized dependency parser, what means that it has been developed and trained to learn POS dependencies, instead of word dependencies. Although their creators have demonstrated quite good comparative results when using this kind of parser [33], they have noticed that context dependent lexicalized parsers usually get better results on their own context. One must notice that this essentially leads to the discussion: generalization vs. over-fitting. Although a good generalization is desired most of the times, in the case of a competition, where most of the possibilities are known, over-fitting can be desired in order to guarantee the best results. This doesn't mean that, in a future development of the robot would probably be needed a more general solution to serve in a wider range of situations and this is why, after the RoboCup, as will be explained in 6.2, the efforts are being spent on generalizing the current solution.

Malt Parser

There are other state of the art parsers, like MaltParser. In this case, Växjö University and Uppsala University provide only what they call a 'data-driven' dependency parser[34]. This software is essentially an engine that learns the treebank data from a corpus and so it can learn to parse. This has been tested on many languages, but is only provided with the engine and a sample in english. For this reason, using it requires the creation of a dependency treebank, the training of the parser and eventually a long phase of testing. From one side, the fact of learning to parse the sentence on your corpus has the advantage that the parser will be over-fitted to your kind of data, what is desirable given that Stanford Parser, for example, has been already trained with data that is useless or even confusing for our robot. On the other side, the data that we have about the sentences is dependent on the vocabulary, that is given 5 days before the competition starts. Preparing a corpus and editing it and testing within these five days is a dangerous bet.

Also, in this case, there hasn't been any new version of the parser since 2010, so it was considered already unsupported.

4.1.4 Methodology

In order to deal with this first objective, a specific output was defined to communicate the semantic output with the goal of the reasoner. This means to define which is the essential information that is needed for the reasoner to work and, thus, which are the basic elements of a goal to be fully achievable.

In [24] are proposed two approaches to choose from:

- Train the speech recognition in a wide range of users
- Keep speaker-independence and be able to recognize a smaller range of sentences

In this thesis the second one has been chosen because the sentences are more or less predefined and the range keeps considerably small.

Looking at the example phrases and the RoboCup rules, one can observe that four is a complete and sufficient number of the parameters that are relevant to define a goal, although not all of them are needed in every case:

- Which **action** is needed to perform
- Which **location** is relevant for the given action
- Which **item** is relevant for the given action
- Which **person** is relevant for the given action

An *imperative* sentence denotes explicitly the desire of the speaker to perform a certain *action*. This *action* is always represented by a verb. Although a verb may convey an occurrence or a state of being, as in *become* or *exist*, in the case of imperative sentences or commands the verb must be an action and if it wasn't,forgetting that a command of this kind would be semantically questionable, it wouldn't be achievable by our robot, given that it is as it has been presented and in this context, it is impossible for the robot to change of state or condition. Also in the context of RoboCup it's important to notice that all the commands convey an action (go, bring, give, introduce, follow, etc).

Knowing this, it's known that any command will ask the robot to do something and these actions might be performed involving a certain object (grasp a coke), location (navigate to the kitchen table) or a person (bring me a drink). Not all the verbs might be accompanied by these three elements and it is known beforehand that a command won't involve more than one item or person at the same time. *'Bring a coke to John and a fanta to Mary'* is not allowed, while it's perfectly plausible to listen *'Get me a drink from the fridge'*.

After receiving a sentence from the ASR, it becomes easy, then, to identify which is the action. First of all, the sentence is tokenized. This, in Natural Language Processing means to separate the sentence in terms, what in this case means separate the sentence by its blank spaces (the reader should remember that there is no punctuation marks on the output of the ASR), what also means into words. After this, NLTK toolkit and also the Stanford Dependency Parser includes already trained Part-Of-Speech tagging functions for english that work reasonably well. This kind of function complements each of the previous *tokens* with *tags* that describe which is the POS more plausible for each word. Applying POS-tagging two important achievements are obtained:

1. The verbs are immediately found and so, the most important information is obtained: the *action*. One should notice that finding three verbs in category one commands in fact mean that three commands are given, which must be accomplished correlatively.

2. The given command might be decomposed in sub-commands in the first category cases, given that each verb in the sentence stands for exactly one specific command that can be considered individually if the order is preserved (together with the co-references). Given that there is no punctuation, it seems a reasonable way to find the three sentences in the first category cases.

Until this point the action that is needed to eventually perform has been already extracted. The only thing that is left to obtain the complements of that action. One can realize two approaches to obtain this information.

- The first, and most simple, consists on identifying from all the nouns in a sentence, which words are already objects, persons or locations, looking inside an ontology. The strong point of this approach is that it is very simple to construct and to keep up to date, but the problems come when you have adjectives, appositions or nouns that belong to more than one category. Another inconvenient could come with a long complex sentence where a noun could be referring to many verbs, although this case is not in the scope of this project.

- The second approach consists on finding the dependencies between the words in the sentence. This makes the system more robust: having a dependency tree allows to see which parts of the sentence are connected to each other and, if the case, which connectors do they have. This means that by finding a dependency tree, i.e. with the Stanford Parser, allows to find which noun acts as a direct object of a verb and looking for the direct object, you should find the item over which the action should be directed. The same happens with the indirect object or even locative adverbials, which tell us the function of the related words in order to extract the relevant information that was mentioned above. Obviously this approach has also some drawbacks, the most important of them is the fact that constructing a dependency tree is:

  - Not trivial
  - Slow

  This means that although it provides information more useful and a solution more general, it's not very robust and can lead to unexpected errors, specially when the solution can be ambiguous and the sentence become more complex. Another more specific drawback is that there is not much research on parsing sentences in imperative mood.

A third approach, which is under development, is based on the use of *verbNet* [28], a state of the art, on-line lexicon that, with the proper tools can enforce the dependencies that could find a dependency parser using information retrieval techniques. Using an interface that communicated both *VerbNet* and an ontology like *WordNet* [35] would lead to software more robust and probably capable of dealing with erroneous sentences.

Although this, the chosen technique is a combination of the first two. The construction of a Context Free Grammar for the special context of the RoboCup that covered a wide range of sentences like the ones presented before, from both, category I and II. The reason why the CFG was constructed is that Stanford Parser, although being promising, wasn't trained to handle this kind of sentences and easily got tangled with the long sentences in category I. This parsing was faster than Stanford's, but the drawback is that it is very restricted to RoboCup-like sentences and could not cover some other verbs or different kind of complements. It's easy to see now that having a lexicon like VerbNet could automatically enhance the coverage of the hand-made grammar by adding near 4000 extra verbs and contexts, while WordNet would allow to check the consistency of the dependencies by checking whether a word in a, i.e. locative adverbial is a location or an object, what is actually made by a hand-made ontology.

4.2 Reasoning and Planning: SOAR

Referring the second objective presented in section 2.3, some kind of cognitive architecture like those described in section 3.2. In this specific case, SOAR has been selected for the reasons that will be explained after a brief evaluation of other cognitive architectures and reasoning systems. Following this part there will be a brief description of the interface on ROS between the previously developed parser and the cognitive architecture SOAR. Finally, the third point that will be covered in this section is the methodology followed to achieve the main and secondary objectives.

4.2.1 Cognitive Architectures

CRAM

CRAM, the Cognitive Robot Abstract Machine was developed by the Technical University of Munich as a software toolbox for enabling design, implementation and development of cognition-enabled autonomous robots performing everyday activities. The project was later adopted by the University of Bremen which had support for the PR2 Beta Program (Robot seen in section 3.1.1). Currently it has several modules in ROS, what doesn't happen with ACT-R neither SOAR, what makes it an easy-to-use toolbox for any robot working under this 'OS'.

The idea of CRAM was born from the investigation in autonomous robot control systems that enable robots to perform complex everyday manipulation activities in human living environments. There exist a number of middle-ware software libraries that support the development of distributed modular control systems, like ROS or Orocos [2, 36].

Cognitive mechanisms are needed because a robot which performs everyday manipulation tasks must continually decide which actions to perform and how to perform them [37], such as learning, knowledge processing, and action planning. This are higher level capabilities that are not implemented in the previous middle-ware libraries, which are more centred on the low-level control.

CRAM language includes, not only the previously mentioned cognitive capabilities, but also data structures, primitive control structures, tools and libraries specially designed to enable mobile manipulation and control. The CRAM toolbox facilitates the implementation of this complex control programs that reason about the world to fulfil specific high-level goals.

In order to complete this toolbox, CRAM for Everyday Manipulation (CRAM-EM) includes a set of extension modules that help robots in everyday manipulation tasks as recognizing and localizing objects, reaching good locations for manipulation and grasping those objects. They also have a data base of knowledge called KNOWROB [38], which helps as a common sense ontology, providing CRAM the knowledge needed for taking decisions. The representation on KNOWROB is action-centred and is allowed with uncertainty management and automatic acquisition of knowledge. In KNOWROB there are many modules including general service robot knowledge for manipulation of objects at home and kitchen environments.

The goal of this project wasn't building on the robot something with a robust functionality already done by others, but providing the system with the capacity to reason and decide by itself, while CRAM is more like a toolbox for cognitive reasoning specially based on pre-defined recipes, where others have more straightforward methods for human-like reasoning, learning and deducting. From this point of view, the project must be more on the development of a cognitive module for Reem rather than changing the classical state-machines based control system into another language. In fact, CRAM, more than a cognitive architecture is a cognitive enabling tool, and this difference stands for the fact that instead of full reasoning, CRAM holds a lot of pre-defined, expert based knowledge and uses it to plan for solvable goals. The reasoning concept in CRAM stands for choosing the next good enough action to perform (saying the best one would be far too optimist and beyond the reality) in order to achieve the current goal, while unified cognitive architectures like SOAR and ACT-R are able to learn from impasses and generate new behaviours.

SS-RICS

Symbolic and Sub-Symbolic Robotic Intelligence Control Systems (SS-RICS) is tool intended to embed the state of the art theories of human cognition and understanding of the human mind into robots, to say, a sort of 'Robotic Cognition Theory' [22].

SS-RICS was developed under 5 principles:

1. The lowest level of perception includes algorithms running in a parallel fashion, while the highest levels of cognition are algorithms operating in a serial fashion.
2. At both the low levels and the high levels of cognition, the algorithms are relatively simple. It is the interaction, processing, and results of simple algorithms which produce complex, intelligent behaviour.
3. Pre-programming SS-RICS should be guided by the algorithms that have been developed through evolution. The pre-programming that is done should allow for the emergence of complex behaviour, but not be the complex behaviour itself.

4. Cognitive development within SS-RICS is principally about the reorganization of memory elements through increasing and decreasing their respective strengths.
5. Cognitive development and change can occur after a given amount of time or after the necessary low level elements are in place to allow for higher level processing.

The last two principles describe how cognitive change takes place: through either the strengthening or weakening of memory elements or through changes in goal directed behaviours after certain low level systems have fully developed.

Although it seems a very interesting approach, specially for a service robot that will have to eventually perform a wide range of actions, this system was dispatched due to its lack of robustness, and overcomplexity in the task solving. It takes control of every action, while in our case, REEM already have the grasping, object recognition, face recognition, navigation, etc. macro-functions already implemented on the core of its behaviour.

Re-learning these core capabilities joint with the fact that I didn't had full-time access to the robot and PAL Robotics was not interested on re-doing the work already done were the clue reasons why it wasn't selected for this thesis.

So, our case of study is contradictory to the third principle of the development of SS-RICS, because it is based in the macro-actions that are provided with the robot, while low-level control is not possible. The cognitive characteristics in principles four and five stand in detriment of this thesis developmental and time constraints: learning by the activation of enough memory elements does not ensure that certain high-level processing can be raised on the current environment. Must be noticed that the object of this thesis resides in controlling the robot by the high-level reasoning on a low-featured, simplified world.

SOAR

SOAR is a cognitive architecture that has been under continuous development since the early 1980s [39]. Since Newell defined SOAR as his view of a *Unified Theory of Cognition* [40] a lot of researchers have been developing agents with SOAR to emulate cognitive abilities from humans, specially in visual recognition [41], games Wintermute2007, Wintermute2007a, Mohan2009, emotional behaviour [42, 43], etc.

The characteristics that make of SOAR a cognitive architecture for general intelligence[1]. The aim of Soar has been to search for a minimal set of mechanisms that are sufficient to

[1]Definition taken from [44]

realize the complete range of general behaviour[45]. The principal mechanisms of Soar architecture, relevant to this thesis, are described below:

Parallel, associative memory Soar uses an extremely efficient symbolic pattern matcher that compares the representation of current context to the activation conditions of each element in the system. Given that the associations of this comparison are made independently one from each other they occur parallel in soar. Thanks to this parallelism, all the relevant knowledge is considered on a problem so a Soar agent could simultaneously consider different approaches to solve it.

Belief maintenance To allow Soar agents to exist on dynamic environments, Soar uses a computationally inexpensive truth maintenance algorithm that updates beliefs to respond to changes in the environment automatically without having to create extra knowledge to manage these changes.

Preference-based deliberation Deliberation is needed to give more importance to certain beliefs, what allows to achieve goals. This deliberation is made in Soar by explicit representations of preferences, so using the current knowledge the agent can reach a decision, from multiple options.

Automatic subgoaling In the case that the Soar agent couldn't decide between options, soar recognizes this conflict, marks it as an impasse and automatically creates a subgoal to solve this impasse. This allows the agent to create new problem contexts and plan, compare different possible situations and come up with an action that helps to solve the impasse.

Decomposition via problem sub-spaces The agent can also be in a situation where it knows what to do but it doesn't have an immediate function to achieve the goal. This would generate another impasse in Soar. Automatically, Soar generates a new goal to find a way to achieve that goal. This mechanism allows for the decomposition of a goal into subgoals, what allows narrowing the number of considerations into a small set of choices.

Adaptation via generalization of experience After an agent solves an impasse, coming up with a decision, soar summarizes and generalizes the reasoning made during the impasse to create a new piece of knowledge that will allow to make front to new similar situations. These new rules are called chunks.

ACT-R

ACT-R is a theory of the mind that borns to shape a different approach to that of Newell in [46] with SOAR. He argued for cognitive architectures that would explain how all the components of the mind worked to produce coherent cognition. In his book, he described the Soar system, which was his best hypothesis about the architecture. ACT-R is a cognitive architecture called for adaptive control of thought–rational [47] and is another hypothesis of this kind of architectures.

ACT-R implies that declarative knowledge is fairly a direct encoding of things in our environment and procedural knowledge is a fairly direct encoding of observed transformations, as in SOAR. The combination of both by encoding the statistics of knowledge use in the environment is its basis of learning and, so, the main difference with SOAR[48]. While SOAR finds a path to the goal and creates a rule to encode this learnt path, ACT-R uses the statistics of different executions to generate Bayesian rules.

The decision then between ACT-R and SOAR resides on the usability of the architecture. From one side, the language that both architectures is very similar. Their approach is based in if-then clauses, written in a Lisp-like language. On the other side, SOAR have a more adaptive language thanks to the fact that several actions and conditions can be added in a single rule. Besides, ACT-R fire and check the rules sequentially while SOAR makes this in a parallel-like manner. Having to add the rule firing and memory access in a buffer, forces to consider bottlenecks when programming ACT-R rules. SOAR parallelism is emulated in order to ensure that all changes in memory are done at the same time and independently from the ones fired before. On conflict resolution, SOAR is centred on operator preferences one over the other, while ACT-R prefers sub-symbolic metrics as utility or activation to define the preferences. A problem comes given the sequentiality of ACT-R, the preferences are generated dependent of the previous results. To get a better insight on this differences one can look at [49, 50].

In this case, there is no special preference for each of them in the context that they are being applied given that the reasoning will take place in strictly high-level features. So, thanks to the ability to match several memory elements with a single rule, that makes the construction of the model easier and the parallelism of SOAR, that allows for more intuitive development and avoids the consideration of bottlenecks, SOAR has been selected as a slightly better option.

4.2.2 Why SOAR?

There are many reasons for which would be desirable to have architectures like CRAM or SS-RICS. The most important is that both of them are robot-oriented toolboxes that provide them with more or less extensive cognitive abilities. The interesting point is that both can reason in a sub-symbolic level, at which they can achieve robot control in terms of, i.e. object manipulation or computer vision, in addition to symbolic reasoning, which stands for the kind of reasoning that is performed directly with discrete objects or symbols. A symbol here is understood as in [51]:

'Synonymous substitutes for longer expressions that are required frequently'

In this case, these longer expressions correspond to specific actions which are already implemented on our robot or even behaviours. Specially in this case, a symbolic representation, also known as high-level representation, is interesting because it can be seen as a simplified representation of the world and the robot capabilities in such a way that it becomes easier to think and reason over this simplified vision than in the continuous, full-featured world that we, as persons, are interacting with everyday.

Also, Reem has already most of the capabilities encoded in state machines under ROS[2], joint with all the controllers and sensing abilities, that experts have been and currently are enhancing and optimizing to obtain the best from the robot, everyday. Also this code, belonging to a private company is not fully available to the students, so it's needed to work using the available state machines or, sometimes, building or editing some in order to achieve the desired behaviour of the robot.

This specially means that the manufacturer of Reem gives no support for the implementation of algorithms that allow for the control of the robot, like CRAM or SS-RICS. Also these algorithms are useful if the objective was to learn to grasp a can or to learn to identify a beer bottle, but are not so useful if one want only to encode the actions of the robot and the conditions of the world and, given a goal, perform or learn how to perform the variety of tasks that the robot needs to accomplish during the General Purpose Service Robot tests.

For this reason SOAR seems a good choice, given that the other programs are a bit more control oriented. Specially they follow the paradigms stated in [22, 37, 48, 52] of a sensor based cognitive architecture, what is not fully integrable on our robot. For this reason is only and strongly needed the development of the reasoning module in a symbolic basis.

Additionally, SOAR provides three important characteristics that the others doesn't:

- The subgoaling capacity that is explained in [44], that allows Reem to find a way to get out of an impasse with the current actions available in order to achieve the desired state, as it would be the case in which the robot couldn't decide the best action in the current situation with the available knowledge.

[2]As it is done, although in a completely different context, in [20]

- The chunking ability, that was first described by Newell, Laird and Rossenbloom in [44] and easily explained and summarized in [53] and [45], ability that allows the production of new rules that help de robot to adapt to new situations and, given a small set of primitive actions, execute full featured and specific goals never faced before.
- Reinforcement learning [54], that together with the two previous features helps the robot to learn to perform maintained goals such as keeping a room clean or to learn using user-defined heuristics in order to achieve not only good results as with chunking, but near-optimal performances.

This means that SOAR provides us with an architecture where one can:

- Represent all the necessary knowledge symbolically[3].
- Include firing rules, encoding constraints and requirements to execute each available action.
- Represent desired states.
- Generate non pre-defined rules in order to achieve a desired state, given a set of actions and the available knowledge about the surrounding world. [4]

Still, Soar is not a planning system and therefore it's not as well-suited as a pure planner [55].

4.2.3 Interfacing SOAR

Reem runs under the platform ROS, that was described in section 2.2.2. But neither NLTK nor SOAR have been used under this platform before and, so, there's no support and no implementation exists already. Luckily, NLTK is a toolkit written in python and so, given that ROS is supported also in python [2], python has been chosen as the main language for these project.

So, why this interface is needed?

1. SOAR is executed under Java
2. Some kind of connection or communication is needed to obtain the information of the ASR,

[3]All the knowledge that can be represented symbolically.

[4]One should notice that this last characteristic is not necessary if the actions are encoded correctly.

3. To send the parsed knowledge to SOAR,
4. And to call the actions desired from the robot by SOAR
5. Everything must be executable by the robot itself, automatically, without human intervention

With this list in mind, the interface was developed in a modular fashion as ROS developers encourage. The first module described will be the speech recognition module, that is started with the robot, waits for a command and converts it into a message published in the second module. This second module is a ROS service that waits for a command message, adapts it to be interpreted by SOAR, hopefully fulfilled and then returns a callback to allow the robot to continue with a new command.

Speech Recognition Module

ROS provides a package that allows the structure of the robot functions into state machines that can, at the same time, become states for other state machines. This modular feature has been used for all the tests in RoboCup and, although this specific test is aimed to escape this rigid way of programming the robot, small state machines are still built to take care of some specific functionalities that have a more rigid and closed working schema. For this reason, a state machine was constructed in charge of:

1. **Introduction** The robot speaks and introduces herself.
2. **Check ASR** Checks that the speech recognition module is running.
3. **Ask for a new command** The robot asks for a new command.
4. **Listen the command** The robot activates the Speech recognition, processes the command and returns a sentence with the text understood. This state is provided by the company, although was slightly modified to accept the grammatics and vocabulary specific for the GPSR tests.
5. **Parse the command** The parser receives as an input the raw text sentence and outputs the set of [Action, Item, Person, Location], as explained in the previous section 4.1.4.
6. **Ask for confirmation** The robot says the information that he extracted waiting for confirmation that it is correct[5].

[5]Notice that this is the processed information already.

7. Publish The state machine finally publishes the obtained knowledge in a ROS Service node. It halts until receive the callback message confirming that the robot can continue asking for a new command at step 3.

This state machine is launched at the setting up.

SOAR Service Module

Services are the way provided by ROS to communicate between nodes. Running a service on a node, allows it to receive messages from subscribed publishers and send answers back to subscribed ones. Also it can perform some process inside, an this is the characteristic that makes them interesting to communicate with SOAR.

Given that this module interfaces SOAR, it becomes more complicated than the previous one and needs a bit of explanation of how the SOAR program has been implemented, although the next section will explain this point with finer detail.

The SOAR program, as explained in 4.2.1 is rule-based and descriptive. The organization of the reasoning within SOAR can be simplified as the following loop:

1: Generation If the agent exists, then generate all the knowledge about the world.

2: Proposal If there's any operator that fulfil its firing conditions, then propose that operator.

3: Preferences If there is any reason for which an operator should change its preference value, then change it.

4: Application If there is one best proposal, then select that proposal, ignore all the others and apply its Right-Hand Side (RHS) actions. If not, generate an Impasse. In any case, return to **2: Proposal**.

Knowing this working sequence, SOAR is given:

- A set of rules to propose actions
- A set of rules that change the world after an action is executed (with *aborted* or *succeeded* outcomes)
- A set of modifiers to the action proposal preferences[6]

[6]At this point, must be explained that there are to versions of the program. The one presented here, finished and another one under development. This last version differs in the fact that it doesn't have this preference modifiers and take profit of the chunking process and will be detailed in the last section of this thesis.

- A rule that initialize the *world*[7] and generates the current desired state[8].
- A rule that checks that the current state matches the desired state and returns a success outcome.

Deeper detail is given in the following section.

For this reason, the following points must be covered:

- How the world is initialized?
- How do we generate the goal?
- How is the world updated?
- How are handled more than one goal (as in Category I sentences)?

This questions are answered in this section.

How the world is initialized?

There are many ways to cover this point. One can simply generate the world setting in a SOAR interpretable file, that can be stored as a template. The other is to generate the memory elements directly in the interface and input them to SOAR, on demand. Due to time restrictions to present the program during the RoboCup@Home competition, the first one was chosen and implemented because it is more robust[9] and more simple[10], so it was a better option to reach the deadlines. [check that this sentence is correct]

Although this, the second option was implemented after the RoboCup in order to improve the functionality of this SOAR implementation.

How do we generate the goal?

The goal, given that it is a reduced version of a *world* is generated in the same format as the world is initialized, with the exception that given that the goal depends strongly on the message that comes from the parser, it can not only be filled but must be compiled from the (at most) four pieces of knowledge that are provided. This is because the effect

[7]The *world* here means the knowledge about the environment of the robot. In the context of RoboCup, means exactly the persons, items and locations that the robot needs to know about, and its current situation (not location).

[8]This desired state corresponds to a set of *world* elements that need to be in a certain state. The elements not mentioned in this object are ignored.

[9]Once you have written the program that generates the file, you always call the desired functions right before running SOAR and the world will be generated as the first action.

[10]generating a text file in python, given a template consists only on filling gaps or replacing labels, while constructing a whole structure inside SOAR's input-link and defining working rules that copy this structure in the correct place, robustly is a bit more complex

on the world is different depending on the action: the state of a coke is not the same when you deliver it or when you grasp it.

For this reason, a simple compiler that, knowing the desired action, translates it into a desired state and, then, converts it into a SOAR's desire that it can interpret and fulfil. To achieve this easily, the same world structure inside SOAR has been designed in python objects with similar attributes and the necessary functions, so they can be worked with easily and the translation is almost immediate.

How is the world updated?

Having two representations of the *world*, one inside SOAR and the other in the interface requires the *world* to be updated regularly. To be more precise, this is chosen to be after each action, because is when the robot interacts with it that it gets changed[11].

This means that after the action resolves, it returns to SOAR an object describing the success or the failure of the action and the relevant changes it provoked, where this information is used to change the current knowledge of the robot. I.e. if the robot has detected a beer bottle and her next action is to grasp it, it will send the command 'grasp.item = beer_bottle', while the action response after resolving should only be a 'succeeded' or 'aborted'[12] message that is interpreted in SOAR as 'robot.object = beer_bottle' or simply not changing anything in the *world*.

On the other side, in the interface, the representation of the *world* is only updated after the goal is achieved (or not), with the final state obtained. This allows the new incoming goals to be generated in the current context and not in an obsolete one. [13]

How are handled more than one goal (as in Category I sentences)?

In the presented version of the interface, a sentence of category I is split into 3 different goals and sent like so to the interfacing service. This way, in order to preserve the execution order, the three commands are converted into a list and one goal at a time is taken from the list and sent to SOAR. This means that, in the first implementation of this software, the one sent to the RoboCup@Home competition of this year, each goal was being printed directly on the init file of our SOAR program as the new desired state, together with the updated *world*. It's clear that this is not the best option, because creating a new agent each time (although it preserved the previous knowledge) is not

[11]Obviously, this must be understood in the context of RoboCup@Home, where there are no unexpected events.

[12]Considering a good performance of the object recognition module.

[13]Here again, its important to refer to section 6.2, because this keeps only for the first version of the interface. At this moment, all the updating is being done inside the SOAR architecture, because new goals are directly asked by SOAR and not published in a service, but this version is still being debugged and, so, under development.

as efficient as keeping the same agent alive for all the test, but time restriction made necessary to work in other areas knowing that this approach is coarse although robust enough for the expected performance.

Answered these questions, the service that interfaces the parser with SOAR simply reads the published messages or what is the same, the commands, and generates a *world* based on that information. Must be considered that this *world*, for simplicity, contains only the persons and objects involved in the command, so each time the robot receives a command, she only knows what she should interact with, and nothing else, except for the locations, which she can access all of them. This isolates Reem, so it can only interact with the world to accomplish her tasks. It's important to know that, in any case, the software is considering every item and person for the recognition of commands, but restricts this information to the robot in purpose.

The service also compiles the first goal in the stack and generates the initialization file of our agent, so it can execute SOAR and update the world when the goal is achieved.

As mentioned before, SOAR runs on JAVA, while our interface, parser and also the ROS platform are programmed in Python. Hopefully, SOAR includes an interface that allows a user to use most of its features, using SWIG[14] on Python. Using this, an agent is created in SOAR with the knowledge mentioned above, so it can begin to reason and choose an action to perform. When the action is chosen, it is printed in the output link of the agent and is read in the interface, that calls the specific function, i.e. navigate; with some parameters, i.e. the location is the kitchen table. This function is a state machine already embedded on the robot as most of the robot capabilities already are. SOAR waits for the action to finish and then receives the outcome of the state machine execution, and continues until the desired state is achieved.

The main advantage of having an interface like this is that, in the end, the reasoning will be independent on the platform that it is running on. If there are the functions that perform the desired actions (grasp, find object, navigate, recognize faces, learn faces, etc.), they only need to be imported in the interface and the SOAR agent will do all the reasoning.

4.2.4 Implementation

Within SOAR, the reasoning agent has been implemented in four main parts:

[14]Simplified Wrapper and Interface Generator, allows to easily use libraries from a programming language to another

Initialization

First of all, when the agent is created, the best and only operator proposal is to initialize the world. This is a common practice in SOAR, given that when an agent is created it has no Working Memory Elements (WME) and they must be generated somehow. So it is proposed to generate a *world* and a desired state if there is a state without an attribute 'name'. This first world will contain:

Robot a Robot has an ID [1][15]; it is in a location (locId), known [0-?] or unknown [-1]; may have an object (obj1Id) [0-?] or not [-1]; it may have pointed to a location(pointedAtLoc) [0-?] or not [-1]; and it may have introduced herself already [yes/no].

Object An object is more extense than the robot because there are many more modes of interaction within him.

- **id** It's the name of the object. Here is a numeric identification that is translated into the correct word outside of SOAR. For the agent, the name of the object is irrelevant if it is distinct from the others, because the interesting characteristics will be encoded within the other attributes. It can get values from 0 to the total number of objects. A value of -1 would indicate that the object is not identified.
- **locId** Indicates the current position of the object. Again, it is an identification number that will be translated when required outside the reasoner. If the location of the object is known it will get a value greater than 0, while an object which localization is not known yet will have a value of -1.
- **near** Indicates if the object is near, so it can be grasped. The object is considered near if it is in the same location as the robot. Given that locations are tables or other furniture, the same location Id, in an object means on the location, while in the case of the robot is next to the location, but facing it, ready to find objects, correct the position and grasp. It can get yes or no values.
- **toBeGrasped** This attribute means that the object is graspable in the sense that the robot could grasp it if it was in range. It can get yes or no values.
- **grasped** This other attribute stands to indicate if the object was once grasped or not. It doesn't change if the object is delivered in a position or not. It's the characteristic in the robot which indicates if there is an object grasped in her hand or not. It can get yes or no values.

[15]The robot object represents the robot herself, so it will always be only one robot that represents her current state.

- **delivered** Denotes if the object was once delivered or not. It can get yes or no values.
- **found** Indicates if this object has been already found (the object detection has found it, independently of knowing its location) or not. One should see that these last three attributes are only important to check specific commands for accomplishment. It can get yes or no values.

Person Persons are like objects and there will be only one person available for the robot if he must interact with it. If this person's name is unknown, its Id will be -1 to indicate this fact.

- **id** It's the name of the person. This numeric Id is translated into the name of the person. An unknown person has an Id with value -1, which will change if an action like 'learn-person' is executed successfully.
- **locId** Indicates the current position of the person. Again, it is an identification number that will be translated when required outside the reasoner. If the location of the person is known it will get a value greater than 0, while a person which localization is not known yet will have a value of -1.
- **near** Indicates if the person is near the robot. This attribute have the same meaning as in the case of objects. It can get yes or no values.
- **obj1Id** As in the case of robots, a non '-1' value in this attribute indicates that the person is holding the object with this Id, or, what is equivalent for the robot, that this object was delivered to this person.
- **askedName** Either it has a name or not, this field indicates if the person's face has been learnt. It can get yes or no values.
- **found** Indicates if this person has been already found or not, as in the case of objects. It can get yes or no values.
- **followed** Denotes if a successful execution of the action 'follow' has taken place. It can get yes or no values.
- **recognized** This field shows if the person was recognized with a face recognition algorithm or not. It can get yes or no values.

location Locations only have an Id field and are available only to have all the places where robots, persons or objects can navigate, be, or be placed on.

During the initialization phase, all this objects that can be needed are generated as described in the init file, joint with the current goal, that is, in the end, a short representation of what has been described above. While the *world* contains fully featured elements[16], the goals only mention the specific characteristics of the *world* that they need to be accomplished, i.e. if you need a coke delivered to John, it isn't specified in the goal where should be John or wether the coke was found or not. More specific information about goals will be detailed in the end of this chapter.

Actions

The set of actions that the robot can performed are encoded as operators. This means that there is, for each possible action:

- A rule that proposes the operator, with the corresponding name and attributes.
- A rule that, if the operator is accepted, sends the command through the output-link.
- One or several rules that depending on the command response, fire and generate the necessary changes in the *world*.

Given the nature of the SOAR architecture, all the proposals will be treated at the same time and will be compared in terms of preferences. If one is best than the others, this one is the only operator that will execute and a new deliberation phase will begin with all the new available data. It's important to know that all the rules that match the conditions are treated as if they fired at the same time in parallel, there is no sequential order [21].

The set of actions that have been considered are the following:

navigate Action The robot navigates to a certain location.

Firing Rules This rule is proposed if the execution exists. For robustness, the rule is also proposed as best if any other action needed to fulfil the current goal requires the robot in a location different from her current location.

introduce-me Action The robot introduces herself.

Firing Rules This rule is proposed only if a person is near, given that it is illogic to introduce yourself to no-body. It is propose as the best option if the goal states expressly that the robot must be introduced.

search-person Action The robot scans her proximity to see if there is a person near. This function does not look for specific persons, just finds if there is one near with the laser sensors.

[16]All the elements have all the fields filled in the *world*.

Firing Rules This rule is proposed always, although it is only proposed as the best option if there is a person, related to the goal with an unknown position (-1).

memorize-person **Action** The algorithm to learn a person's face is launched. Once the face is learnt, the name is asked and, so, the Id of this name is set to the person that is near.

Firing Rules There must be a person near with an unknown Id to be proposed and it is always proposed as the best option in this case.

recognize-person **Action** The face recognition algorithm in the robot is launched to recognize a person in front of the robot.

Firing Rules The rule fires only if there's a person near that is already known, and confirms its identity. [17]

search-object **Action** Reem looks for a specific object in the current location.

Firing Rules The rule is proposed when the robot is in the same location as an object. If the current goal has something to do with an object, it will be proposed as the best action, because the only interactions available with objects can be to find, grasp or deliver them.

grasp **Action** Reem tries to grasp an object near him.

Firing Rules If the robot is near an object already found and she doesn't have any object at hand, the grasp operator is proposed, and is proposed for the best operator if the goal needs some interaction with this object.

deliver **Action** The robot offers an object to a person.

Firing Rules This operator is proposed only if there is an object held by the robot and there's also a person near. If the goal's name is deliver, the person is the correct one and the object in the goals corresponds with the one grasped by the robot, it is proposed as the best operator. The deliver action can be done in locations instead of delivering to persons, but the working schema is exactly the same.

point-at **Action** The robot approaches to a designated location and points at it.

Firing Rules This is only proposed when the goal is to point at, because if not, the robot chooses to point at objects randomly if she doesn't know which option is better to do.

[17]Although the common sense order would be: first attempting the recognition, if it fails, try to learn the face, in RoboCup these orders are always together, and first they make the robot learn a person's face and then recognize him, in order to check that the robot truly learnt the face. For this reason, the firing conditions of this function are proposed like this.

Most of this actions were already encoded in the robot, but some of them had to be adapted for the GPSR tests. The needed capabilities were originally programmed to work in specific environments or other tests in RoboCup@Home. For this reason, some parts of bigger state machines were selected and re-used in this test in order to be interfaced with SOAR and perform adequately.

Elaborations

Parallel to this, involving one or several actions, a special kind of rule, that is called *elaboration* is in charge of making some changes. This special rules are in charge of the preferences of each proposal, checking what operator is better than which other and which is the best. They won't be detailed here because they are only pieces of code that make possible the correct reasoning for the robot.

This elaborations might be created by two different ways:

- The first of them is, using experts or common sense, encoding all the knowledge possible about which logic should use the reasoning process. In example:
  - There is a rule to preferences navigation to a certain location against any other action if the location of a person or object in the goal is not the same as the location of the robot.
  - There are elaborations for each operator to clean the input and output links after their information has been used.
  - The rule that checks that a goal has been achieved is in fact an operator that checks that there isn't any condition that doesn't match.
- The second option consists on using the learning ability of Soar. If it is configured to handle subgoaling and chunking, Soar can generate the necessary rules to achieve the goal, developing preferences for specific operators under certain conditions.

The first option is the one used in this thesis, and a combination of both is being used in the next version of this work. This is because it allows for less restricted behaviours, adapting to new goals never faced before, while keeping some custom-made restrictions that one could consider that shouldn't be forgotten.

Goals

The goal in Soar has been described many times in previous sections but this section is centred in the descriptions of the process of deciding when the goal has been achieved.

First of all, this goals are projections of the received commands in Soar language. As explained before, a command is translated into an object that has the same elements as the *world*, but represents a future world, more precisely, a *'desired state'*. Given that this *desired state* is a representation of the *world*, it can contain persons, objects, locations and robots, but it has also a name that matches the received command.

Representing the desired state as a complete world, would require the definition of all the elements with all its attributes and matching everything to consider the goal satisfied would be unnecessary because you usually want one specific element to get into one specific state, while everything else is completely irrelevant for your purposes, so, if you want a coke delivered to John, you don't want the robot to consider anything about Mary or if the robot introduced herself besides delivering the right object to the right person.

For this reason, All the goals are the most simplified version that was thought of, like delivering only requires a specific object to be in a specific person's hand, the object to exist and the robot to exist. This means that when checking the accomplishment of the goal, you will need to find if at least the set of elements exist and match the required attributes.

Matching rules in Soar is made by comparison. Soar checks if there is a specific element with a specific attribute with a specific value. If it is true, the rule fires. Dealing with the problem in this way would mean that one should generate the goal checking rule every time a new goal is created and this is not a fast approach. Trying to generalize in one single rule something that changes in number of elements to compare, attributes and values is not feasible with specific values and generating a rule for each goal promised to bring problems when adding new features and goals to the system[18] and would make the program slower, having to check every time for more rules.

The alternative was making the goal checking more general. If you have any element in the *world*, with some attribute matching the value of the same element and attribute in the *desired state*, and this held for each element and attribute in the desire, the desire would be achieved. But this rule cannot be implemented directly because there is no 'for all' function: Soar only checks if there's one instance holding this, and this instance might not be the same in two different instances of the rule. To achieve this, instead of checking that the goal was achieved, a goal achieved proposal is always proposed as the best option and it is set into the worst one if there is any element with a given Id and any attribute in the desire and the same element with the same Id and the same attribute in the world that has a different value or has no value.

[18]The addition of a new goal would mean that a new rule to deal with this goal would need to be compiled when the command was received.

Due to the simplicity of the language that uses Soar, this implementation was not obvious.

Chapter 5

Results Evaluation

5.1 Results

The best test bench to evaluate the results of this work is the participation in RoboCup@Home League. There, the settlement is controlled by the organizers, but unknown to the participants, so the results are the best evaluation possible.

But first of all, I'll present some results and evaluation of the experiments performed in the laboratory.

5.1.1 Semantic Extraction

The semantic extraction, as explained in the previous section 4.1, was finally performed using a Context Free Grammar using the Natural Language ToolKit (NLTK). Before selecting this options, strong efforts were put in using the Stanford parser. Given that in 2011 was improved with better recognition of imperative sentences[56] and the extensive background that this parser has, it was considered a good option to begin with the semantic extraction considering a good result of the parser. The problem began when the complex phrases of *Category I* came into play. Stanford parser presumes to be an unlexicalized dependency parser[56]. This means that, although being probabilistic, it is not specialized in a specific domain, because instead of using lexical and morphological information to extract the dependency tree, it uses only Parts of Speech tagging to train the parsing[33]. From this, one could see that this kind of parsing can be trained with more data and is more general, what leads to certain difference in the maximum achievable accuracy with lexicalized parsers, which, correctly trained on their specific domain, can achieve arround 90% , while unlexicalized parsers are still near 86% [33]. This is important, because having long sentences without punctuation marks leaves place to errors and having three verbs in a sentence is not the most common case in normal texts. With this, Stanford parser was producing around three different dependency trees for each possible sentence due to ambiguities, most of them, over complex because it is more usual to find subordinate phrases instead of coordinate ones. Together, this lead to have less than a 25% of the sentences correctly parsed, while all the others needed some heuristic to choose the correct parse tree. Having three phrases to check, per sentence, seemed more practicable to create from scratch an appropriate CFG rather to get stuck in trying to use a software that wasn't design for this purpose. The use of a CFG have two main advantages and two main flaws:

- It is easy to design (having studied morfosintaxis in high-school) and implement.
- It is custom made, specific to the desired domain and provides the desired information, not more, not less.
- It is restricted to work only in the designated domain. It will always fail in the cases not contemplated during the design.
- Modifying it, to say, adding a new command that could be needed to parse, may imply that the full CFG should be changed and adapted, what implies more work in the future.

Although this, given that the context is closed and specified to the RoboCup@Home, this option is more simple, effective and robust than the Stanford Parser. Also, it showed an accuracy in the dependency parsing of the 100%.

The rest of the semantic extraction is dependent on the dependency tree, so having the CFG working perfectly ensures that the semantics are correctly extracted.

5.1.2 Reasoning

Analysing the results of a reasoning system is almost as hard as defining the concept of *intelligence*. Many researchers may center its results on the aspects on which they are interested, although there will probably be other aspects of the reasoning process that are controversially dropped. In my opinion, given that we are testing the reasoner on a service robot, the following questions are those more important to see if the reasoning module was or not a success, although they are always aspects of the most important features of any problem:

Robustness This implies answering two questions: Does the reasoner find situation where it can't decide any more? Does the reasoner call actions that lead to the automatic failure in the accomplishment of the goal? How does the robot handle this situations?

Completion The robot should have available actions to handle any kind of situation to which it is exposed, what means having all the necessary primitive macro-actions to fulfil any possible command and also being able to reason how to achieve this desired state, for all the possible goals.

Speed The robot shouldn't spend more time thinking than acting. To be more precise, a service robot should be able to avoid situations in which it is stopped, reasoning or processing any information.

Scalability The problem presented in this thesis, the one at RoboCup@Home, is a simplified version of the tasks that a service robot could need to handle in a real-life environment. Although it would be the selling company who defines the specifications of its robot, it is interesting for the company to have the possibility to expand its functions easily and that the reasoning still works after this expansions.

Robustness

In terms of robustness, the current version of the program doesn't usually crashes, because for the RoboCup@Home League was necessary that the robot wouldn't stop at the middle of a task and had to be rebooted. For this reason, the reasoning was restricted to work consistently and no freedom was let to the robot. With this, the actions that the robot decided to perform were chosen because there was always a best option to choose at every moment. The problem is that in the case that the robot fails the execution it will try it again, what, in the worst case, will mean that the robot will by trying the same action spending all the allowed time in the test. To cope with this, the actions that are more likely to fail, ask for help in the case that they abort for some reason, for example, if the robot can't find a way to grasp a drink, it will ask for the drink to be put on her hand or if she can't find a person, it will ask to the person that comes in front of her. Although this is not the best outcome, it allows the robot to continue with her goal, instead of getting stuck in one action for the rest of the time.

Completion

In this case, the system developed for this thesis, guarantee that the actions proposed will lead to the goal, so the robot will find a solution, although it can't be assured to be the optimal one. For each command that the referees can ask, there is a set of actions that allow its completion and Soar have the means to choose the right actions in the right order to fulfil that command. Although this, Soar, and so the scope of this work, are centred on reasoning about primitive actions or, more specifically, pre-defined macro actions, which, as it was described in section 4.2.4, are encoded in big, static state-machines. This means that, although they have been tested, they also can fail sometimes and the failure might lead to a state of no return, i.e. if a grasp is two centimetres too far from the bottle and it falls because of the collision. This is not controlled by Soar and cannot be accessed by a student, so this situation would led to the impossibility to finish the current goal, although it is the worst case and obviously PAL Robotics is working on making the robot more robust and dynamic.

Speed and Scalability

Finally, speed and scalability are so tied together that will be evaluated simultaneously in this section.

First of all, in the current state of the system, it can reason about 11 different goals[1], using 10 different abilities. The amount of productions already in use is of 77 rules at the beginning, that will be checked at every moment. In terms of speed, this amount isn't significative and the scalability would only imply adding new rules for new actions or goals. Hopefully, only 4-5 productions are usually needed to add a new ability in Soar, but the rules under which this new action should or could be called have to be defined by an expert. Given the amount of variability that allow our world (how and where are the objects and persons, and their amount) as more possibilities are introduced in Soar, more difficult is to define the rules in such a way that they work on every possible case. In the next chapter, will be proposed to add the subgoaling feature that has Soar, in order to, having all the actions defined, you can send possible goals in a simulated environment and Soar can learn how to achieve these goals. The problem comes when the number of different problems learnt approaches one hundred. Then, the amount of learnt rules increases with the addition of new goals, what increases the execution time in an unpredictable way[2]. Although this, in [50, 57] some techniques to face this problem, called the utility problem, are proposed in order to reduce the number of chunks checked, from which, in specific applications [57] show that only a 10-25% of the chunks are used 3 or more times.

[1]This 11 goals are a simplification of the 23 verbs that are currently supported by the parser and can be somehow synonyms in the context.

[2]The number of rules created by chunking depends on the definition of the actions and the specificity of the goals.

5.1.3 RoboCup@Home

Although the concrete and detailed results presented in the previous two sections, Reem was also presented in the RoboCup@Home League and the results and feedback obtained from there must be presented and evaluated.

After several successful trial with different orders of *category II* in the laboratory, where the robot was able to understand the sentence correctly, retrieve the knowledge by voice, provide it to Soar, which provided all the times a correct sequence of actions that fulfilled the command and returned to the referee, the robot was sent to Eindhoven, where RoboCup 2013 was taking place.

There, several technical issues were found and tried to solve temporarily for the event, related to grasping, navigation and speech recognition. Specially the last, was caused because the speakers of the commentators where pointing to the arena instead of the public.

Although this, Reem was presented in front of the referees. There, she was given two commands from category 2, which were recognized correctly. The parser identified the actions commanded, which were to carry a drink to a table and to bring a snack from an appliance. After this, the robot tried to obtain the kind of drink/snack desired and the exact location of those items, but it couldn't understand the last words.

Must be considered that only two teams out of 21 successfully understood the commands, while only one could retrieve the missing information. The other one was our team. After retrieving the information, the team renounced because they weren't able to execute the command, but only understand it.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This work is the first implementation of Soar in a robot that participates in the RoboCup@Home contest. It is also the first time that this robot, Reem, is able to receive orders in natural language and accomplish them. Despite some issues with the calibration of the grasping and the speech recognition, this implementation has shown to be promising for the next participation in the RoboCup league. More precisely, with a bit more of work, Soar could be in charge of solving most of the challenges in RoboCup@Home, given that it already have most of the necessary abilities.

The dependency parsing has proven to be enough to extract the semantic information of commands, although using a Context Free Grammar is a non scalable option that should be changed to another approach. Stanford Parser could work for this purpose if the sentence was split in *category I* cases, given that *category II* was working all the times. Given the format of the sentences, probably is not a good idea to define an heuristic to choose the best tree, because there is a large variety of sentences and if sometimes the context was extended to a real environment, the heuristic might be reviewed. Another option is the use of a wrapper using VerbNet and WordNet. A correct development of this wrapper would provide the roles of each element in the sentence more easily for a wider variety of sentences. Again, the combination with WordNet - that has been already done in [58], although this hasn't been applied in real cases - would allow to understand the errors in the sentences of *category III* of the GPSR, that had semantic errors[1].

[1]This would be done by checking that the category of the noun in a complement, matches the category of the complement of the verb itself.

The simplification of the reasoning process into primitive macro-actions has demonstrated to have a major problem in the cases that this macro-actions are not robust enough. For this reason, the macro-actions must be tested for a long time and in a variety of situations, so the reasoner can trust in the outcome of the action, meaning that it won't stop moving too far from a table or it will never throw a bottle on the floor. Once this is achieved, we can trust in the actions of the robot, we can predict them and, so, model them correctly in Soar to make the reasoning more stable. Besides, the reasoning has proven to be strongly robust and, under the previous assumption, allows for very fast response from the robot in a variety of situations. It would allow high-level reasoning, leaving the control, detection or other specific functions to be better designed by experts on those software.

Another important thing, related to the future work in next section, is the fact that using chunking, the rules to perform any task would be automatically generated to solve it under the current state, so the next time that a similar situation occurred, the rules would fire and the problem would be solved without having to plan the task again. This could allow, not only to fulfil the goals in GPSR, but also in most of the other tests that the robot has to do during the RoboCup@Home, like the Clean up, The Cocktail Party or the Restaurant tests, given that the actions needed for these tests are the same as in GPSR and only the goal should be defined.

6.2 Future Work

Finally, the hypothesis that I raise is that a robot running under an adequate cognitive architecture can eventually become a good service robot even with not so good set of actions, if they are simple enough. One first thread of future work begins on this line. Based on the initial statements of Soar and ACT-R [44, 48], the cognition architecture should be simple to allow for the intelligent behaviour to happen. If the actions to reason about are too high-level, the reasoning is constrained to the restrictions of this piece of software. Would be interesting, although probably not feasible with Reem, to define a set of even more simple actions, implement them in Soar and let the program learn. The drawback of this approach is that the benefit of using macro-actions is lost. The trade-off resides in wether use high-level reasoning for unparametrized fast planning or to control the whole robot using Soar.

The second line of work, and in fact, the thread that I am developing right now and have been mentioned several times in this thesis is adding first chunking and later reinforcement learning to the system created. This implies a lot of new additions that were intended to be included in this thesis but that, in the end, are still under development. First of all, adding subgoaling and chunking to the current systems involve several changes:

- Removing the current elaborations that modify the preferences of the operators to allow the subgoaling assign its own preferences.
- Removing the service and launching Soar directly in a node. This is not intuitive, but this allows the Soar agent to keep alive for an undefined period of time and, so, learn and let us learn how it works.
- Including two new operators: the first one to ask for new commands and the second one to ask for the missing information. This implies that..
- ..is needed to change the goal compiler to convert it in the format of Soar Working Memory Elements.
- The current operators must be modified to be proposed always that the action could be executed, independently of the programmers knowledge of how to perform the task in the current set of actions.

This points would allow to perform some experiments that theoretically should work well. Specially that of providing the same goals that the robot has been already solved with the current version of the software and that we know that are achievable. After this, would be very interesting to provide new goals to the robot, for which the definition of the other tests in RoboCup@Home would be a beginning. I must say that all this previous points are already implemented and being tested and debugged, but are not yet in a stable version.

Next important improvement is the addition of reinforcement learning on the system. This would stand to achieve maintained goals, like cleaning up a room, that needs that the robot continuously checks the room to find things out of their place and puts them on their right location.

Bibliography

[1] Pal Robotics. http://pal-robotics.com/.

[2] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS : an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, 2009.

[3] Tamás Haidegger, Marcos Barreto, Paulo Gonçalves, Maki K. Habib, Sampath Kumar Veera Ragavan, Howard Li, Alberto Vaccarella, Roberta Perrone, and Edson Prestes. Applied ontologies and standards for service robots. *Robotics and Autonomous Systems*, pages 1–9, June 2013. ISSN 09218890. doi: 10.1016/j.robot.2013.05.008. URL http://linkinghub.elsevier.com/retrieve/pii/S092188901300105X.

[4] Http://www.ifr.org/service-robots/. Internation Federation of Robotics (IFR), service robots.

[5] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *IEEE Internationa Conference on Robotics and Automations*, pages 1321–1326, 1998.

[6] Y. Ogura, H. Aikawa, K. Shimomura, H. Kondo, A. Morishima, H.-O. Lim, and A. Takanishi. Development of a new humanoid robot WABIAN-2. In *IEEE International Conference on Robotics and Automations*, pages 76–81, 2006.

[7] K. Akachi, K. Kaneko, N. Kanehira, S. Ota, G. Miyamori, M. Hirata, S. Kajita, and F. Kanehiro. Development of Humanoid Robot HRP-3P. In *IEEE-RAS Internationa Conference on Humanoid Robots*, pages 50–55, 2005.

[8] Shuuji Kajita, Kenji Kaneko, Fumio Kaneiro, Kensuke Harada, Mitsuharu Morisawa, Shin'ichiro Nakaoka, Kanako Miura, Kiyoshi Fujiwara, Ee Sian Neo, Isao Hara, Kazuhito Yokoi, and Hirohisa Hirukawa. Cybernetic Human HRP-4C: A Humanoid Robot with Human-Like Proportions. In *Robotics Research, The 14th International Symposium ISRR*, pages 301–314. 2011.

[9] M. Gienger, K. Loffler, and F. Pfeiffer. Towards the design of a biped jogging robot. In *IEEE International Conference on Robotics and Automations*, pages 4140–4145, 2001.

[10] I.-W. Park, J.-Y. Kim, J. Lee, and J.-H. Oh. Online free walking trajectory generation for biped humanoid robot KHR-3(HUBO). In *IEEE Internationa Conference on Robotics and Automations*, pages 1231–1236, 2006.

[11] Young-Su Cha, KangGeon Kim, Ji-Yong Lee, Joongjae Lee, Minjun Choi, Mun-Ho Jeong, ChangHwan Kim, Bum-Jae You, and Sang-Rok Oh. MAHRU-M: A mobile humanoid robot platform based on a dual-network control system and coordinated task execution. *Robotics and Autonomous Systems*, 59(6):354–366, June 2011. ISSN 09218890. doi: 10.1016/j.robot.2011.01.003. URL http://linkinghub.elsevier.com/retrieve/pii/S0921889011000108.

[12] Michiel van Osch, Debjyoti Bera, Kees van Hee, Yvonne Koks, and Henk Zeegers. Tele-operated service robots: ROSE. *Automation in Construction*, August 2013. ISSN 09265805. doi: 10.1016/j.autcon.2013.06.009. URL http://linkinghub.elsevier.com/retrieve/pii/S0926580513001064.

[13] Thomas Buschmann, Sebastian Lohmeier, and Heinz Ulbrich. Humanoid robot Lola: design and walking control. *Journal of physiology, Paris*, 103(3-5):141–8, 2009. ISSN 1769-7115. doi: 10.1016/j.jphysparis.2009.07.008. URL http://www.ncbi.nlm.nih.gov/pubmed/19665558.

[14] S Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, June 1999. ISSN 1879-307X. URL http://www.ncbi.nlm.nih.gov/pubmed/10354577.

[15] Hanafiah Yussof. Biped Locomotion of a 21-DOF Humanoid Robot for Application in Real Environment. *Procedia Engineering*, 41(Iris):1566–1572, January 2012. ISSN 18777058. doi: 10.1016/j.proeng.2012.07.351. URL http://linkinghub.elsevier.com/retrieve/pii/S1877705812027518.

[16] Jeong-Ki Yoo, Bum-Joo Lee, and Jong-Hwan Kim. Recent progress and development of the humanoid robot HanSaRam. *Robotics and Autonomous Systems*, 57(10):973–981, October 2009. ISSN 09218890. doi: 10.1016/j.robot.2009.07.012. URL http://linkinghub.elsevier.com/retrieve/pii/S0921889009001031.

[17] Shiwali Mohan and John E Laird. Situated Comprehension of Imperative Sentences in Embodied , Cognitive Agents. 2012.

[18] John E Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive Robotics using the Soar Cognitive Architecture. In *Proc. of the 6th Int. Conf.on Cognitive Modelling*, pages 226–230, 2004.

[19] Scott D. Hanford. *A Cognitive Robotic System Based on Soar*. PhD thesis, 2011.

[20] Sam Wintermute, Joseph Xu, and James Irizarry. SORTS : Integrating SOAR with a Real-Time Strategy Game. Technical Report January, 2007.

[21] Sam Wintermute, Joseph Xu, and John E Laird. SORTS : A Human-Level Approach to Real-Time Strategy AI. pages 55–60, 2007.

[22] Troy Dale Kelley. Developing a Psychologically Inspired Cognitive Architecture for Robotic Control : The Symbolic and Subsymbolic Robotic Intelligence Control System. *Internation Journal of Advanced Robotic Systems*, 3(3):219–222, 2006.

[23] Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie. Developing High-level Cognitive Functions for Service Robots. *AAMAS '10 Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 1:989–996, 2010.

[24] Rehj Cantrell, Matthias Scheutz, Paul Schermerhorn, and Xuan Wu. Robust spoken instruction understanding for HRI. *Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction - HRI '10*, 1:275, 2010. doi: 10.1145/1734454.1734555. URL http://portal.acm.org/citation.cfm?doid=1734454.1734555.

[25] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. *Proceeding of the 5th ACM/IEEE international conference on Human-robot interaction - HRI '10*, page 259, 2010. doi: 10.1145/1734454.1734553. URL http://portal.acm.org/citation.cfm?doid=1734454.1734553.

[26] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[27] Paul Kingsbury and Martha Palmer. From TreeBank to PropBank. In *In Language Resources and Evaluation*, pages 1989–1993. 2002.

[28] Martha Palmer, Karin Kipper, Anna Korhonen, and Neville Ryant. Extensive Classifications of English verbs. In *Proceedings of the 12th EURALEX International Congress*, 2006.

[29] Daniel Gildea and Julia Hockenmaier. Identifying Semantic Roles Using Combinatory Categorial Grammar. In *EMNLP '03 Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 57–64, 2003.

[30] Donghyun Choi and Key-sun Choi. Automatic Relation Triple Extraction by Dependency Parse Tree Traversing.

[31] Steven Bird. NLTK : The Natural Language Toolkit. *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics.*, pages 1–4, 2005.

[32] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with Compositional Vector Grammars. 2013.

[33] Dan Klein and Christopher D Manning. Accurate Unlexicalized Parsing. *ACL '03 Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, 1:423–430, 2003.

[34] Johan Hall. *MaltParser – An Architecture for Inductive Labeled Dependency Parsing.* PhD thesis, 2006.

[35] Christiane Fellbaum. *WordNet: An Electronic Lexical Database.* 1998.

[36] R. Smits, T. D. Laet, K. Claes, P. Soetens, J. D. Schutter, and H. Bruyninckx. Orocos: A software framework for complex sensor- driven robot tasks. *IEEE Robotics and Automation Magazine*, 2008.

[37] Michael Beetz, M Lorenz, and Moritz Tenorth. CRAM — A C ognitive R obot A bstract M achine for Everyday Manipulation in Human Environments. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[38] M. Beetz and M. Tenorth. KnowRob — Knowledge Processing for Autonomous Personal Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009.

[39] Pat Langley, John E. Laird, and Seth Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, June 2009. ISSN 13890417. doi: 10.1016/j.cogsys.2006.07.004. URL http://linkinghub.elsevier.com/retrieve/pii/S1389041708000557.

[40] Allen Newell. SOAR as a unified theory of cognition: Issues and explanations. *Behavioral and Brain Sciences*, 15(03):464–492, 1992.

[41] Samuel Wintermute and Scott D Lathrop. AI and Mental Imagery. 2008.

[42] Eric Chown, Randolph M Jones, Amy E Henninger, Green Ct, and Ann Arbor. An Architecture for Emotional Decision-Making Agents. In *AAMAS '02 Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 352–353, 2002. ISBN 1581134800.

[43] Amy E Henninger, D Ph, Ann Arbor, and Randolph M Jones. Behaviors that Emerge from Emotion and Cognition : A First Evaluation. In *AAMAS '03 Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, number 1985, pages 321–328, 2003.

[44] John E Laird, Allen Newell, and Paul S Rosenbloom. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 1987.

[45] SoarTechnology. Soar : A Functional Approach to General Intelligence. Technical report, 2002.

[46] Richard L Lewis, Scott B Huffman, Bonnie E John, John E Laird, Allen Newell, and Shirley G Tessler. Soar as a Unified Theory of Cognition : Spring 1990. 1990.

[47] John R. Anderson, Dan Bothell, Christian Lebiere, and Michael Matessa. An Integrated Theory of List Memory. *Journal of Memory and Language*, 38(4): 341–380, May 1998. ISSN 0749596X. doi: 10.1006/jmla.1997.2553. URL `http://linkinghub.elsevier.com/retrieve/pii/S0749596X97925535`.

[48] John R. Anderson. ACT: A Simple Theory of Complex Cognition. *American Psychologist*, 1995.

[49] Randolph M Jones, C. Lebiere, and Jacob A Crossman. Comparing Modeling Idioms in ACT-R and Soar. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, 2007.

[50] William G Kennedy and J Gregory Trafton. Long-Term Symbolic Learning in Soar and ACT-R. *Cognitive Systems Research*, 8(3):237–247, 2007.

[51] Hugh MacColl. Symbolic Reasoning. *I see Mind*, 1880.

[52] Changyun Wei and Koen V Hindriks. An Agent-Based Cognitive Robot Architecture. pages 54–71, 2013.

[53] Andrew Howes and Richard M Young. The Role of Cognitive Architecture in Modelling the User : Soar ' s Learning Mechanism. (01222), 1996.

[54] Shelley Nason and John E Laird. Soar-RL : Integrating Reinforcement Learning with Soar. In *Cognitive Systems Research*, pages 51–59. 2004.

[55] A Stenger, B Fernando, and M Heni. AUTONOMOUS MISSION PLANNING FOR UAVS : A COGNITIVE APPROACH. In *Deutscher Luft- und Raumfahrtkongress*, pages 1–10, 2012.

[56] University of Stanford. http://nlp.stanford.edu/software/lex-parser.shtml#Download.

[57] William G Kennedy, Ceralene Drive, and Kenneth A De Jong. Characteristics of Long-term Learning in Soar and its Application to the Utility Problem. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

[58] Maria Teresa Pazienza, Marco Pennacchiotti, Fabio Massimo Zanzotto, and Via Politecnico. Mixing WordNet , VerbNet and PropBank for studying verb relations. 2006.